Presenting student project in written form: A guideline

Dr Zine-Eddine Bouras,
Department of Software Engineering,
Faculty of Information Technology,
Philadelphia University, Jordan.
E-mail: zbouras@philadelphia.edu.jo

January 2009

Aims:

To introduce the skills needed to present a student project effectively in written form.

Learning objectives:

- Understand how to structure and write professional reports.
- Write clear and concise abstracts.
- Understand how to present data and results clearly.
- Understand how to reference material and avoid plagiarism.
- Document software, comment programs and write user guides.

***This guideline is inspired from the book "Projects in Computing and Information Systems. A Student's Guide" of C. Dawson, Pearson Education Limited 2005.

1. Introduction

Dissemination of your ideas and results was identified as an important part of the research process. Quite often the report is the only evidence of the project when it is finished, unless student developed a substantial piece of software (and even then people may only get to read student report rather than use the software). As the report represents student project, remember that the good work you have performed can be ruined by a poor report. There is no point in performing a tremendous amount of valuable and important computing work, research and development, if you cannot present your findings to other people. However, a bad project cannot be turned into a good one by producing a good report. Although you can improve a poor project with a good report, you must remember that your report is a reflection of your project and you cannot disguise sloppy investigation, development, implementation, analyses and method with a few carefully chosen words.
This report focuses on the presentation of written material for your project: structuring reports, writing abstracts, referencing material and presenting data. It also covers topics such as documenting software, commenting programs and writing user guides.

2. Writing and structuring reports

2.1. Consideration

There are two main considerations that you should bear in mind as you begin work on your project's report:

a. **What is the purpose of the report?**

- Is it to obtain the best mark you can achieve for your project?
- Is it to present your work in the best light?
- Is it to disseminate your ideas and results to others?
- Is it to provide a thorough literature review of the field?
- Is it to inspire others and to persuade them to get involved with your research?
- Is it to fulfill the requirements of your course?

b. **Who is going to read it?**

- What do they already know?
- What do you want them to learn?
- What do you want them to gain from your report?
- How do you want to influence them?
- Will it be read by people other than your examiners (future employers, other students, academics and experts, for example)?

These considerations will influence what you decide to include in your final report and also the style of writing and presentation that you adopt (for example, technically oriented text for experts or more explanations and examples for the more novice reader). You should not include material merely for the sake of it as this might irritate the reader and appear like 'padding'. Similarly, you should not leave material out of your report if you think it is important for the targeted readership. Try to get the balance right understand what it is you are trying to say, be aware of what the reader already knows, and include material appropriately.
Department of Software Engineering of Philadelphia University has guidelines on how reports should be presented - for example, layout of the title page, word counts (upper **and** lower limits), font size and type, line spacing, binding, the number of copies of your dissertation/report that you should submit and so on. It is important that you follow these regulations as failure to do so may mean that your report is rejected.

2.2 Approaches to writing

There are two main approaches that people tend to use when they write reports: the top-down approach and the evolutionary delivery. These two approaches are not mutually exclusive and you may well find yourself adopting both of them to one extent or another as you develop your report or dissertation.

The top-down approach is used to identify the structure of the report with a chapter breakdown structure - how many chapters it will have, what each chapter will contain and how each chapter will break down into sub-sections. With sub-headings identified you can then go on to complete these sections at an appropriate point in your project when results are obtained and information is acquired.

Identifying the content of the chapter as a number of component parts makes writing much easier and less daunting as individual sections can be tackled one at a time. Identifying the overall structure of a chapter also allows you to keep an eye on the overall target of that chapter so that you do not discuss extraneous ideas that are out of context with the chapter's main theme (the chapter breakdown structure will probably help you to identify a more appropriate place to enter the misplaced text). Chapter breakdowns also help with time management in that they provide you with a better understanding of the amount of writing you have to do. This stems from an understanding of the complexity of each section, which will give you an idea of how long those sections will take to complete.

You might try to identify sections and sub-sections early in your project. However, as is often the case, it is not until you finally come to completing your project that you fully understand what you want to include and can identify the specific content of every chapter. Whatever happens, you will find that a report breakdown structure is a useful way of arranging your thoughts and ideas and identifying how they link together within the content of your report.

The other approach is the evolutionary delivery. Many people use this approach but are not conscious that they are doing so. In this approach, you begin to write parts of your report and rewrite these parts as your project progresses. Each part thus evolves and matures over a period

of time as new ideas emerge and your understanding increases. You do not sit down at the end of your project and write your report as a one-off. You write it over a much longer period of time throughout your project.

The two approaches introduced above can be combined so that you identify, perhaps at the start of your project, the specific sections of some of your report's chapters. You can then begin to write these sections but will find that they evolve and change as your project progresses. You might also find that your report breakdown structure itself evolves over time as your understanding increases, your ideas change and develop, and you obtain your results.

2.3. When should I start writing?

Since personal computers became commonplace from the 1980s onwards, students no longer leave the final writing up of their projects to the very end. Before this time, students would spend, say, 80% of their effort working on the project itself (the research, development, experiments, etc) before typing up their report on a typewriter. These days, however, it is possible to start work on your report when you start your project. This does not mean that you should start writing sections of the report during the first week, but you should at least be considering how your report will be structured and laid out using a word processing package. It doesn't mean that you will not have a final 'writing up' stage in your project. It will mean, however, that this stage is no longer a write-up of the entire project, but possibly as little as drawing together existing material that you have produced and the completion of, say, conclusions, an abstract and a contents page.

As you work on your project and produce word processed reports, and documentation (for example, requirements specification, design documents, etc) you should try to incorporate these in your evolving report. At the very least, you should be keeping good notes as your project progresses so that when you come to the final write-up, all the material and information you require are readily available.

2.4. The order to writing

There is a particular order to writing that you should try to follow. This order breaks down into the following ten stages:

1. **Identify structure**. This relates to the content of your report, using a report breakdown structure. Although a specific content structure might not be entirely clear to you at an early stage, you should attempt to produce as much detail for each chapter's breakdown as possible. Report breakdown structures were discussed in the previous section.
2. **Identify presentational style**. You should also try to set standards at this stage on the presentational aspects of your report, for example, its layout, font, numbering

conventions, etc. This will save time later when you are trying to collate your chapters and sections and find they are presented inconsistently. Make sure you follow any guidelines that your institution provides.

- Avoid broad, open spaces or cramped layouts. Try to make sure figures and tables do not force large gaps into your text;
- Use a clear 11 or 12 point font. Use something that is easy to read such as Times or Geneva font;
- Use a single, justified column with adequate margins for binding.
- Use page numbers centered at the foot of each page.

3. Draft the introduction. The introduction gives the reader an idea of the report's content, so it should also help you to clarify your own ideas. At this stage, however, your introduction will only be a first draft as your ideas are bound to evolve and your emphasis change by the time you have completed your report. Remember that your introduction might include, or consist mainly of, your literature review. As such, it should be tackled early so that your grounding in the subject is complete.

4. Main body. The main body of your report is the next part you should work on. You might include chapters such as methods used, analyses performed, etc. Clearly the content of the main body of your report will depend on the project you have undertaken. You may find that you write parts of the main body of your report as your project progresses and you will not necessarily write each chapter or section in order. Examples of 'typical' chapters that form the main body of different project reports are presented in the following section.

5. Conclusions and recommendations. Quite clearly your conclusions and recommendations should be one of the last things that you complete. Only when your project is complete you will fully understand what you have achieved and be able to present your final ideas and recommendations.

6. Complete the introduction. As part of the evolutionary approach to writing you may well find that your introduction needs some reworking after you have completed the rest of your project's report. You may want to include some text alluding to your final results or introduce more background on a topic you have since focused on in more detail within your report.

7. Write the abstract. You cannot really write a clear abstract for your report until you know what has been included in it. How to write effective abstracts is covered in detail later in this report.

8. References and appendices. Although you will be collating references and appendices' material as your project progresses, you should not complete their presentation until

the rest of the report has been written. References may be added or deleted and you may decide to include or exclude material from the appendices.

9. **Arrange contents list, index**. Leave the completion of an index (if one is required) and your contents list until the end. Only then you will know the exact content of your report and all page numbers.

10. **Proof-read, check and correct**. It is vitally important to proof-read your report after it is completed. Quite often, because you have been so close to your report for so long, reading through your report straightaway might mean that you miss glaring errors or omissions. You know what you meant to write so this is what you read, whether it is written or not. With this in mind it is a good idea to leave your report for a day or two before proof-reading it or, preferably, get someone else to do it for you. Bear in mind that if you do this you will need to complete your report a few days before its deadline to allow yourself time for proof-reading and correcting or changing any points that emerge.

2.5 Structure

Your report should be structured into the following sections:

- **Title page or cover sheet** - follow any guidelines provided (your supervisor should advise you on this issue).
- **Abstract**
- **Acknowledgements** - to people (and organizations/companies) you wish to thank for helping you with your project.
- **Content listing**
- **List of figures and tables** - this is not compulsory and you should include these lists only if you feel they will add value to your report and will be useful for the reader. Otherwise leave these out as they take time to compile and maintain.
- **The report itself (three main sections):**
    1. **Introduction/literature review**. The first chapter of your report should always be an introduction. Quite often introductory chapters serve to present the literature review. Alternatively, the introduction serves as a brief overview of the project and the report, and the literature review is presented as a chapter in its own right later.
    2. Your **introduction** should set the scene for the project report, include your project's aims and objectives, introduce the project's stakeholders and the topic area, and provide an overview of your report. Also suggest that the introduction should include the 'purpose and situation - why the report was written and what the purpose was'. They go on to suggest that you should

also ‘Target the reader’ - ie you should state clearly at whom the report is aimed - ‘who will be interested in the report’.

3. **Main body** - the content of which depends on the type of project you are undertaking. Some examples are provided below.
4. **Conclusions/recommendations** - summarizes the contribution of the work and identifies future work, etc. This is discussed in more detail below.

- **References** - presented in an appropriate format. Referencing material is discussed in more detail later in this report.
- **Appendices** - labeled as Appendix A, Appendix B, Appendix C, etc. These may include program listings, test results, questionnaire results, interviews you have transcribed, extracts from manuals, letters/correspondence you have received and project details such as your initial proposal, your project plan (and other project management documentation) and meeting reports. You may also include a user manual and installation guide and perhaps extracts or examples of data or data sets used. Consult with your supervisor over what should and what should not be included in the appendices of your report.
- **Glossary of terms** - if required.
- **Index** - if required - but avoid if possible.

A typical structure that many of my undergraduate students use for projects that have involved the development of a software system is:

- **Abstract**
- **Acknowledgements**
- **Content listing**
- **Chapter 1 - Introduction**
- **Chapter 2 - Literature review**
- **Chapter 3 - Requirements**
- **Chapter 4 - Design**
- **Chapter 5 - Implementation and test**
- **Chapter 6 – Evaluation**
- **Chapter 7 - Conclusion**
- **References**
- **Appendices**

The department of Software Engineering has its own development structure that is with the supervisor.

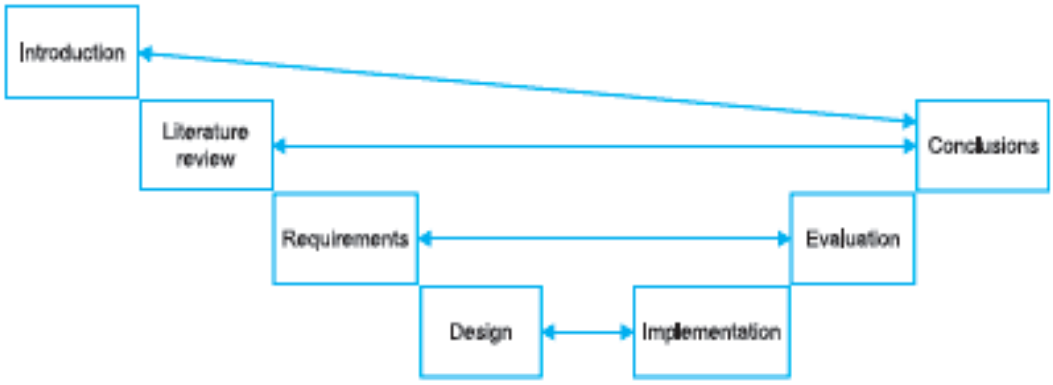

Figure 1. The relationship between chapters

Figure 1 provides an indication of how the chapters in this kind of report structure relate to one another (those of you familiar with software engineering may recognize this as an adaptation of the v-process model). For example, the Conclusions chapter evaluates the project overall - how well it achieves its aims and objectives (outlined in the Introduction) and how it fits in and supports existing work in the field (covered in the Literature review); the Evaluation (of the software) chapter appraises the system developed with the original requirements and evaluates whether those requirements were appropriate; the Implementation chapter discusses how the software was implemented and how the implementation follows the design presented in the previous chapter.
Sometimes students will swap over Chapters 2 and 3 (Literature review and Requirements) to provide the reader with more information about what the project is about before setting it in a wider context. Another variation is to combine Chapters 4 and 5 into a Design and Implementation chapter and move all testing (of the software, including test plans and so on) and evaluation into Chapter 6. Chapter 6 (Evaluation) covers the evaluation of the product developed - ie what the user(s)/client think of the system, how it compares with other systems and so on.

The final chapter (Conclusion) presents an evaluation of the project - it summarizes

What the project has achieved (what has been its contribution)?

How the project has met its initial aims and objectives (and if not, it explains why)?

How does this project fit into and enhance existing work in the field?

This chapter also covers a number of other issues. For example:

Was the development process model used appropriate (if not, why not and what else should have been used)?

Was the programming language suitable?

What problems did the student face and how did they overcome them?

What would the student do next if they had more time?

If they were to do the project again, what would they do differently?

What have they learnt and experienced from doing the project?

How do they recommend the project should be taken forward in the future?

2.5. Style

The style of writing that you adopt to present your report can be discussed from three points of view. First is the actual presentation style of your report - for example, its layout, font size and so on. This kind of style was discussed earlier. Second is the style of grammar that you use within your report. Quite often good reports can be ruined by poor grammar. The author's meaning is unclear as ideas and results are hidden within long complex sentences that include excessive words and jargon. The third point of view is overall content structure and this will be discussed further later.

A good writing style comes with practice - the more you write the easier it becomes. Reading also helps to improve your own writing skills as you learn elements of good practice and identify interesting ways of discussing and presenting arguments. Having said this, there are some simple rules that anyone can follow to improve their writing style for professional reports. Try to write in the third person - in other words, try to avoid using personal pronouns such as I, you, we, my and so on - but make sure that you don't end up producing elaborate, complex sentences just to avoid this. For example, take the following sentence from a student report:

I interviewed seven people to see what they thought of the system.

This could easily be reworded to a less 'chatty' style and without the 'I' to:

Seven people were interviewed to determine their thoughts on the system.

Your supervisor should be able to advise you on this point, and it may be that the nature of your project requires you to use a more personal approach. Keep sentences short and to the point. Avoid making several points within the same sentence. Avoid abbreviations, jargon and slang. Use simple, rather than complex words; the latter is often irritating for the reader, it clouds the meaning of your sentences and is often used to hide your own lack of understanding about the subject which the educated reader will spot. Try to keep your report gender-free - for example, use 's/he' or 'they' rather than 'he'.

It is common practice to present your report in the past tense as the report represents the results of the project which you have completed.

- Avoid jokes and personal asides. Avoid shortened forms such as 'isn't' instead of 'is not' unless you feel that the report will not flow well without these forms. Make sure you know how to use apostrophes. Finally, make sure that you use a spell checker; sloppy spelling puts many reports into a bad light.
- **Avoid** terms like 'clearly' or 'obviously'. You might understand the fact to which you are

referring but it may not be clear to the reader. The reader may also feel they are being 'stupid' if they don't see the point clearly or obviously.

- **Avoid** red flags. These are claims that are your personal opinion rather than accepted facts supported by the literature. For example, "requirements capture is the longest stage of the software development processes". You should make sure that if you include these kinds of claims you support them with either an appropriate reference or a 'caveat word' such as 'often' or 'sometimes'. For example, the previous statement could be reworded as "requirements capture is often the longest stage of the software development processes".

2.6. Word Processing

For computing students it almost goes without saying that the best way to produce your report is with a word processor of one kind or another. These packages are far more effective than type written or hand written work alone. Almost all word processors these days come with dictionaries and thesaurus facilities built in. In addition, many are equipped with equation editors that can help you produce neat equations embedded within your text. Alternatively, equation editors are available that can be used to 'construct' equations before pasting them into your report.

Be careful when using built-in spell checkers. Many are based on American dictionaries and will change words to their American equivalent, for example 'center' instead of 'centre' (or vice versa if you are an American reader!). Spell checkers might also change spelling 'errors' within verbatim quotes you have used from other authors.
Grammar checkers should also be used with caution. What might appear an elegant, well constructed sentence to you might be changed automatically by a grammar checker. However, if you feel that your grammar is weak, these facilities are invaluable.

2.7 Tips

This section on report writing will conclude with a few report writing tips to help you.

- Set deadlines. Your report will take a long time to produce. If you do not set yourself deadlines and stick to them, you will not finish on time. Using a report breakdown structure can help you to plan your time commitments more accurately.
- Write regularly. Find your best time of day for writing and your favourite location. In other words, make sure that you 'write when your mind is fresh' and 'find a regular writing place' (Saunders et al., 2000: 414). People often find they cannot write with distractions or when they are over-tired.
- Create a work rhythm. Once you are underway, keep going. Don't stop to check a reference if the text is flowing, keep going until you reach a natural break.
- Write up sections when they are ready - when they are clear in your mind. This will also save time towards the end of your project when your project write-up might be little more than a collation of your existing text and producing an introduction and

conclusion.

- Stop at a point from which it is easy to restart. It can often take a lot of time to get going again, so try to stop at a natural break in your report, for example, when you have completed an entire section. Trying to pick up from where you left off the previous day or week can be difficult as you might have forgotten what it was you intended to write. If a break in your work is unavoidable, make a note of what you intended to do next so that when you come back to your writing later you can pick up from where you left off more easily.
- Collate all the material you need before starting to write. Breaking your writing flow to search for a reference or visit the library to trace a vital book will not help.

3 Writing abstracts

The function of an abstract is to 'summarize briefly the nature of your research project, its context, how it was carried out, and what its major findings were'. The abstract provides the reader with an overview of your project and is the basis on which many readers will decide whether or not to read your report at all. With this in mind your abstract should be concise (preferably no more than one half page long), clear and interesting.

Your report's abstract should be one of the last things you write, when you actually know what you have achieved and what the content of your report is. Avoid using references in your abstract as the reader will not necessarily wish to search through your report to find them or be familiar with the author(s) you have cited. In addition, avoid using jargon and acronyms – these should be introduced only in the main body of your report.

There are three possible components to an abstract, the inclusion and coverage of which depend on the nature of the report you are producing: *context*, *gap* and *contribution*. The context introduces the topic area in which your project resides; it can include coverage of related topics and issues, and generally sets the scene for the reader so they can comprehend your project's subject area.

The final component covers the contribution or content of your report itself. In other words, what does your report contain that fills the gap you have identified or what your report contains in relation to the context you have discussed.

To get a 'feel' for good and bad abstract presentation pay careful attention to the way others structure the abstracts that you obtain. Take, as examples, the following abstract – based on an artificial neural network approach to predicting software development costs.

Abstract:
One of the major problems with software development projects is that it is extremely difficult to estimate accurately their cost, duration and effort requirements. This invariably leads to problems in project management and control. Part of the problem is that during the early stages

of these projects very little is known about the problem domain and, consequently, initial estimates tend to be best guesses by a project manager.
Artificial neural networks appear well suited to problems of this nature as they can be trained to understand the explicit and inexplicit factors that drive a software project's cost. For this reason, artificial neural networks were investigated as a potential tool to improve software project effort estimation using project data supplied by a software development company. In order to deal with uncertainties that exist in initial project estimates, the concept of neural network simulation was developed and employed. This project discusses this concept and comments on the results that were obtained when artificial neural networks were trained and tested on the data supplied.

Example of Abstract

4 Data presentation

4.1 Introduction

In almost all projects you will have to present data in one format or another – data you obtain from questionnaires or surveys, software test results, algorithm speed trials, etc. While textual presentation of numeric results can often provide a rather 'dry' interpretation of the information gathered, pictures, in the form of graphs and charts, provide a far more pleasing, intuitive and holistic idea of what is going on. A diagram can often simplify quite complex data which could take a paragraph or more to explain.

Although a picture is worth a thousand words, you must ensure that the picture you are painting is the correct one and you are not presenting results in such a way as to hide their true meaning. Remember, when you compile your report, that you must be objective and present your results in a clear and honest way. This section deals with presenting information using charts and tables, gives various examples of some of the most popular charts that are used, and shows some instances where charts are used incorrectly.

4.2 Presenting charts and graphs

All figures and tables that you include within your report should be clearly and uniquely labeled with a number and a short description. The most common method is to label each figure and table using consecutive numbers prefixed by the current chapter number, for example, 'Figure 1.4' refers to figure 4 of the chapter 1.
Note that it is permissible to label a table and a figure with the same number; for example, Table 8.1 and Figure 8.1 refer to two different items within a report. Above all, be consistent and don't change the way in which figures and tables are labelled from one chapter to the next.

4.3 Checklist

Here we present a checklist of points that you should observe when you have completed tables and figures within your report. For both diagrams and tables it is recommend that you should ask yourself the following questions:

- 'Does it have a brief but clear and descriptive title?'
- 'Are the units of measurement clearly stated?'
- 'Are the sources of data used clearly stated?'
- 'Are there notes to explain any abbreviations?'
- 'Have you stated the sample size?'

For diagrams the following checklist of questions are suggested:

- 'Does it have clear axes labels?'
- 'Are bars and their components in the same logical sequence?'
- 'Is more dense shading used for smaller areas?'
- 'Is a key or legend included (where necessary)?'

And for tables:

- 'Does it have clear column and row headings?'
- 'Are columns and rows in a logical sequence?'

4.4 Common mistakes

You should not include figures and tables within your report just for the sake of it. They should be there to support arguments you make within the text and to clarify, in diagrammatical form, data, results and interpretations you are making. This leads to the first common mistake that people sometimes make in using figures and tables – including them unnecessarily.

The second common mistake made when using charts is to use them inappropriately when other charts would present your data in a much clearer light. Although one might be interested in trying to identify the shape of the underlying distribution of degree grades, a bar chart would be more appropriate in this case.

Another common mistake people make when including charts within reports is to scale them incorrectly. Sometimes this is done deliberately to hide the true meaning of the data that are presented. At other times it is done by accident when you are unsure about what your data are trying to tell you or what your data mean.

4.5 Other data presentation

Not only will you be presenting data in the form of charts and graphs but there are others things you might wish to present too – program listings, designs, photographs, diagrams, etc. Some tips that you should bear in mind when presenting these kinds of data include:

- As for charts and graphs, each figure should be uniquely numbered and labeled.
- Try to keep figures and listings to one page. If code listings spread over several pages then you should consider moving the listing to an appendix and include only short extracts (of interesting algorithms/sections) in the main body of the report.
- Consider alternative ways of presenting diagrams. For example, rather than including several figures showing the evolution of a system's interface design, you could include a photograph showing the preliminary sketches and interim designs next to one another.
- Present pseudo code and designs in boxes rather than 'floating' amongst the text –

5. Referencing material and avoiding plagiarism

5.1 Introduction

It is important that you support the work you are presenting within your report by appropriate references. Much of what you present will have been touched on, discussed, written about or covered by other authors in the past. Thus, any arguments that you make within your report, and especially within your literature review, should be justified by referencing previous research. Material is referenced within reports to:

- **Avoid plagiarism**. In other words you do not present other people's ideas, thoughts, words, figures, diagrams, results, etc, without referencing them, in order to make their work look as if it is your own (you thus credit people with their ideas). Plagiarism can be performed accidentally or deliberately but in both cases it is deemed a serious academic offence. This is one reason why you should perform an extensive literature survey – to ensure that you are not merely repeating the work of others.
- **Identify context**. To place your work in context with other recognized publications. This will strengthen your report by showing how it builds and extends the work of others and how it resides within a recognized academic field of study.
- **Support and validate**. To support your own arguments and validate any statements that you make. If you are making certain claims you will have to support these with either research results or references to other authors.
- **Identify sources**. Provide people reading your report with a comprehensive list of related work that they can use to study your topic in more detail or take your work further. By identifying sources clearly, people reading your report will be able to locate the articles you have used.

There are two aspects to referencing. The first aspect to consider is how to use references correctly within the body of your report – in terms of their presentation and appropriateness – called *citing*. The second aspect is how to present these references correctly at the end of your report. Each of these aspects will be dealt with in turn.

5.2 Citing references

Generally, there are two ways to cite references – the *Harvard System* and the *Numeric System* (also called the *Vancouver System*).

However, we will concentrate here on a general overview of a Harvard-type system. Harvard is the better system to use as the numeric system requires each reference to be identified by a unique number which needs updating every time you decide to add or remove a reference from your report. Quite often the numeric system also gives no indication of the author to whom you are referring and the reader has to search through the reference list at the back of your report to find this information. These days, many word processing packages have reference management systems that enable you to maintain and update references within your report quickly and easily.

However, with or without such a system, it is recommended that you use the Harvard style of referencing, which is more flexible and clearer than the numeric approach. The Harvard-type system uses the name of the author(s) and the year of their publication to uniquely identify each reference within a report. For example, take the following extracts from an undergraduate project report:

It is often said that computing is an art not a science (Smith and Jones, 2003: 20)

or

It is often said that computing is an art not a science. This was first suggested by Smith and Jones (2003: 20) who justified their proposition by. . . .

The article by Smith and Jones is identified by the author(s) name(s) and its year of publication. If you are referring to more than one of their publications of the same year you would append letters to the date (a, b, c, etc.) to uniquely identify each article – thus (Smith and Jones, 2003a) (Smith and Jones, 2003b), etc. The page number (20), where the point in question was made, has also been identified. This is common when referencing books, which obviously have many pages, but not when referencing journal articles.

5.3 Listing references

Generally speaking, the best place to list all the references you have used is at the back of your report, as opposed to footnotes at the bottom of pages or lists at the end of each chapter. This provides the reader with a single compendium of all relevant material that they can easily access. Articles you have used are presented under the heading of either *References* or *Bibliography*. References list only those articles that have been referred to within the report itself. A bibliography will list all the articles you have used in your project but are not necessarily referred to in the body of the report.

Bibliographies are useful for the reader in that they identify all material that is relevant for taking your work forward or understanding it in more depth. For taught degree projects and books it might be more appropriate to include a bibliography, but for a research degree it would not. Your supervisor should be able to advise you on which approach to use.

How you present references will depend on the referencing system you are using

– ***Harvard* or *numeric***. Only the Harvard system will be discussed in detail as the numeric system is basically the same. The only difference with the numeric system is that each reference is presented in its numerical order and is presented with its numerical identifier first. For example:

15. Wilson, G. (2002) *The implications of art*, Gower, London.
16. Herbert, K. (2001) *The art of science*, Chapman Hall, Manchester, UK.

In the Harvard system, the use of italics, commas, colons, upper-case letters, abbreviations (such as Vol. for Volume) and brackets may well be dictated by your own institution's 'in house' style or by a variation in the style used (for example, Chicago style). However, Harvard references should always be presented alphabetically with articles by the same author(s) presented chronologically. Examples are:

Books:

Anderson, J. Jones, J.P. and Peterson, K.K.L. (2002) *The implications of science* (2nd Edition), Pitman Publishing, London.

Benjamin, T. (2001) *Computer science made easy*, Arnold, Leeds, UK.

Note that it is not necessary to include terms such as 'Ltd.', 'Inc.' etc for publisher's names as long as the publisher is clearly known from the information presented. The date that is presented represents the date on which that edition of the book was *first* published. This provides an indication of the age of the book, which would not be apparent by referencing a reprint date that could be several years later. The country of publication is also included if it is not clear from the place alone. For example, cities like New York and London do not usually require USA and UK, respectively.

In cases where the author(s) is an editor of the cited work use (ed) or (eds) to denote this. For example:

Anderson, J. Jones, J.P. and Peterson, K.K.L. (eds) (2002) *The implications of science* (2nd Edition), Pitman Publishing, London.
Benjamin, T. (ed) (2001) *Computer science made easy*, Arnold, Leeds, UK.

Journal articles:

Brown, A. and Wesley, C.W. (2005a) 'An investigation of the Hawthorne Effect', *Management Sciences Journal*, Vol 42(1), pp 47–66.
Brown, A. and Wesley, C.W. (2005b) 'Adaptation of genetic algorithms in Hawthorne Analysis', *Management Monthly*, Vol 28(2), pp 21–23.

Notice the use of letters (2005a, 2005b) to uniquely identify these two articles produced by the same authors in the same year.

Web addresses on the Internet:

Gaynor, L. (2003) *Introduction to artificial intelligence*, <http://www.cai.com/ai/1086> (25 July 2004).
International Group on Complex Systems (2002), *Systems analysis*, Minutes of Second Meeting, 12 June 2002, http://www.IGCS.com/Min/two.html (25 July 2002).

References to Internet sites should include the **full** web address, including *http*, etc.

Make sure that you present the title of the page/article/site/author name where appropriate. These references should also include the date on which the site was accessed. Because the Internet is ever changing, these references may become outdated very quickly.

Trade or company publications:

IAEA (2003) *Guidebook on computer techniques in nuclear plants*, Technical Report Series No 27, International Atomic Energy Agency, Russia.

National Environment Research Council (2002) *Computers in hydrology report*, Vol II NERC, London.

Theses:

Hampson, J. (2004) *The effectiveness of AI in calcite modelling*, unpublished PhD thesis, Department of Computing, University of Strathclyde.

Conferences:

Jowitt, J.D. (2005) *Information systems in a progressive society*, Applications of Information Systems XI, Cartwright, R.A. and Laurence, G. (eds), Rowntree Publications, Leeds, UK.

ISAIS (2005), International Symposium on Applications of Information Systems XI, proceedings of an international conference organised by the Society of IS, held London, 12–16 June 2005, Rowntree Publications, Leeds.

The first reference here (Jowitt, 2005) is for an article presented at a conference. The second reference refers to the conference proceedings themselves.

CD-ROM:

Katlen, P. and Rose, P. (2002) *Information systems in the 1990s*, CAROM CD-ROM, Solar Information Systems, London.

Personal communication:

Sometimes people will say something to you that is useful to quote in your report. In these cases there is no physical record of the statement or source so you have to refer to the quote as a 'personal communication'. In your main text you would cite this as the following example shows – (Smith, pers comm) or Smith (pers comm). You would then list this reference using Harvard style as:

Smith, J. (2004) Personal communication, 12 July.

The references presented above are by no means comprehensive and you will undoubtedly come across an article, some data, or some material from an obscure source that is not covered by these examples. However, unless your institution has specific guidelines to follow when

referencing such material, you will have to present the reference in a way that you feel is appropriate. If your supervisor is unable to help you, remember two things. First, the reference should be clear enough in the body of the report so that anyone reading your report knows to which article you are referring and, second, you have provided sufficient information when listing the reference so the reader can trace that article easily if s/he wishes.

6 Documenting software

6.1 Introduction

People sometimes argue that it is more important to get a program's supporting documentation right than the program itself. They argue that it is always possible to mend 'bad' code that is well documented than it is to fix 'good' code that is poorly documented. This emphasizes the importance of good documentation to support all software systems.

The documentation required to support a piece of software can be immense, covering a vast range of issues, from internal commenting of program code, systems analyses and design notes, figures and system documentation, to test plans and user guides. The following items are a list of topics and documentation you might be expected to cover and include in your project to support any software that you produce:

- An introduction/overview – simple introduction to the program, what it does, who is it for?
- Technical solution adopted – what technical solution has been implemented, is this ideal, is there an alternative?
- Design – systems analysis, systems design, human factors, story boards, etc.
- Software engineering information – structure, definition languages and test plans, etc.
- Development approach used – evolutionary delivery, build and fix, etc.
- Problems encountered – bugs, errors, uncompleted sections of code.
- Limitations – what limitations are there to the program; for example, can it only handle files of a certain size? Is it only calculating results to an accuracy of 10%, etc?
- Hardware/software requirements for running the program.
- Next stage. If you were to continue the project or somebody else was to take over from you, which parts of the software should be developed next? Which parts of the program could be enhanced with new features? Are the code, documentation and comments, etc, at a level whereby somebody could take over from you easily in the future?
- Evaluation of the software – how well does it do what it is supposed to do? Does it satisfy the user's needs?
- User guide – written at the right level of detail for the intended user.

Depending on the nature of your project, you will have to present more or less detail in each of these areas. How you complete documentation such as designs, analyses and test plans is beyond the scope of this guideline as it is dependent upon the development process, the methods

employed and the type of project you have undertaken. For example, a pure software development project would require comprehensive analysis diagrams, test plans and system documentation, whereas a project merely developing a piece of code as a vehicle for presenting some ideas would not. The focus here is on commenting programs and writing user guides as these should be included with any piece of code you produce.

6.2 Commenting program code

Commenting program code is dependent on the programming language used (for example, a third or fourth generation language, an object-oriented language, a formal language and so on), the style of code being developed and the requirements of your course and project. Having said this, there are a number of general guidelines you can follow when commenting your code:

- Understand the purpose of the program you are writing. Who is going to use it, maintain or enhance it, mark it? What is their level of knowledge? If you are merely writing a small program for your own use to test out some ideas, you will not need as many comments as a program that is going to be used and enhanced by somebody else.

- Try to ensure you provide the right level of comments within your program – don't over-comment or under-comment and avoid comments on every single line of code. Comments should tell the programmer something that is not clear from the code itself and they are not there to explain the programming language used. For example:

  X: = X + 1; {add 1 to X}

  This is an example of poor commenting; the comment (in brackets { }) tells the reader no more than they can deduce from the code itself (and the variable name could probably be more explicit too).
  Provided you have used suitable variable names and a logical structure for your program then comments should be limited.

- It is advisable to comment each function/procedure/object/block/screen, etc (depending on the language used). This will explain, at the very least, what each component of your program does and may be the required depth of commenting for someone to understand how the program works and is structured.

- Try to make comments stand out from your code so they don't become buried as a mass of text in your program. For example, tab each *in-line* comment (on the same line as a program statement) clear of the code to the right and keep line spaces around *full-line* comments (comments that have one or more lines to themselves).

- Avoid long-winded explanations. Keep comments brief and clear – you are not writing an essay.

- Avoid wasting time producing fancy borders, header styles and so on. Your comments are there to provide understanding and explanation to your program; they are not there to make it look pretty.

- Make sure you include vital information at the start of your program such as author, date, version number, a description of what the program does and, possibly, a brief explanation of how it does it. These comments are often included as *block* comments several lines of in-line comments providing more detailed explanation.

- Try to make sure that you maintain and update program comments as you amend and develop your software. There is little point in keeping outdated comments in your code that refer to much earlier versions of your program.

In summary, it is probably a good idea to get guidance from your supervisor as to the style and level of comments required. Your department may have guidelines on what is expected in the form of program comments and there may be an 'in house' style you have to follow.

6.3 Writing user guides

There has been a lot of research in recent years into user guides, their structure, presentation, content, usability, 'trainability', minimalist training issues and so on, all of which are beyond the intended scope of this book. For the purposes of this book, we are interested in user guides from a narrower perspective in that your guide is not going to be used by the 'masses' but within your own institution as part of your computing project and part of its assessment.

In this context, any user guides you develop are likely to be presented within separate documents to your final report or included within its appendices. How you present user guides is up to you but the longer they are the more sensible it will be to present them as a separate document. Whatever the case, a user guide should provide the user with at least these pieces of information:

- An overview of the software – what does it do, who is it intended for?
- An idea of its hardware requirements – memory requirements, disk space required, additional hardware requirements such as sound cards, platform requirements (PC, Macintosh, Unix etc), operating system requirements, etc.
- How to load/install the software.
- How to start the software.
- How to end and perhaps uninstall the software.
- Details of any known problems and restrictions imposed by the program.

A user manual should satisfy three aims:

- 'to provide practical information about the software when help is not at hand';
- 'to help inexperienced users get started quickly and with least difficulty';
- 'to help experienced users become productive quickly'.

When writing user guides as part of your project you should begin by identifying your target audience. Will you need a comprehensive guide so that complete beginners will be able to understand your software or will a simple overview of its functionality be sufficient as it will only ever be used by your supervisor?

User manuals tend to come in two different forms. First, as ***training manuals*** – where the user is taught how to use the software through a number of examples that build on one another. Second, as ***reference manuals***, whereby experienced users can 'dip into' the manual at appropriate points for clarification/explanation of specific features of the program. How you structure your documentation will be based largely on your intended users. For experienced users, a reference manual may be all that is required. However, for inexperienced users, evolutionary examples may be more appropriate. In addition, depending on the nature of your user, you may have to provide detailed explanations describing simpler operating principles such as 'save as', 'page set-up', etc. It is also a good idea to include some screen dumps from your program in a user guide so the user feels they are following your guide correctly when it appears that things aren't happening as they would expect. It also provides the user with additional confidence to see things mapping out on the screen in the same way they are presented on paper. You might also wish to include a description of possible mistakes that could be made by a user and how the user can avoid or overcome them.

Quite a lot of user guides and help systems are embedded within programs themselves these days. While some of the points made earlier are relevant to these kinds of systems, their integration and technical implementation issues are beyond the scope of this book. You should consult with your supervisor and your client for advice and requirements on this issue.

Reference

C. DAWSON. Projects in Computing and Information Systems. A Student's Guide, Pearson Education Limited 2005

Appendix A:

Research project Template

Approval
Certificate
Dedication
Acknowledgement
Abstract (English and Arabic
Table of content
List of figures
List of table

Chapter 1: Introduction

- Adopted SDLC with justification
- Adopted Analysis and Design methodology – How does it fit the selected SDLC?
- Simple project plan to show project framework

Chapter 2: Software Requirements

2.1 Problem Statement
2.2 Non-Functional Requirements
2.3 Functional Requirements
    2.3.1 Actors identification
    2.3.2 Use case identification
    2.3.3 Use case diagram
2.4 Software Requirement Verification

Chapter 3: Software Analysis

3.1 Class model
3.2 State model (for significant classes)
3.3 Activity model (for significant classes)
3.4 Sequence/Collaboration model (for significant use cases)
3.5 Software Design Verification

Chapter 4: Design

4.1 Design Constraints
4.2 From Analysis to Design (map)
4.3 Interface Design
4.4 Data Design, Architectural Design
4.5 Component based design (program)
4.6 Design patterns

Chapter 5: Coding

5.1 Traceability. (All design blocks must have a class which implements it).

5.2 Automatic Code Generation from Design. (Use tools translating design to code).

5.3 Interface programming. (How your interface objects are programmed. An example).

5.4 Controls programming. (How your Control objects are programmed. An example).

5.5 Entity programming. (How your Entity objects objects are programmed. An example).

5.6 Subsystem programming. (How your packages are programmed. An example).

5.7 Data access programming. (How you store and retrieve data. An example.)

5.8 Concurrency programming (centralized and distributed). (How your processes and their control are implemented. An example).

5.9 Programming with errors avoidance. (How you have used errors avoidance techniques for implementing your system. Examples).

5.10 Programming with/for reuse. (Have you developed reusable components or reused one. Examples).

5.11 Fault tolerance programming (for critical software. If your system is critical, how have you programmed its tolerance to faults).

Chapter 6: Software Testing

6.1 Functional Test Cases

6.2 Sample of effective results

Conclusion and Future works

Conclusion, future works, and Evaluation including critical appraisal for the work done.

References

Appendix